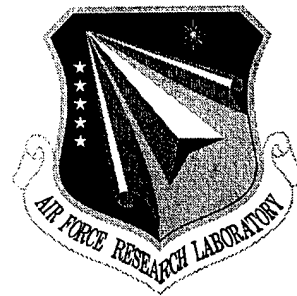


**AFRL-IF-RS-TR-2001-67**  
**Final Technical Report**  
**April 2001**



# **UPDATE ANALYSIS IMPLEMENTATION (UAI) ADVANCED TECHNICAL PROTOTYPE (ATP)**

**PRC, Inc.**

**J. Huff and S. Ochsner**

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*

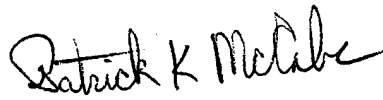
**20010607 022**

**AIR FORCE RESEARCH LABORATORY  
INFORMATION DIRECTORATE  
ROME RESEARCH SITE  
ROME, NEW YORK**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2001-67 has been reviewed and is approved for publication.

APPROVED:



PATRICK K. MCCABE  
Project Engineer

FOR THE DIRECTOR:



JOSEPH CAMERA, Chief  
Information & Intelligence Exploitation Division  
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFED, 32 Brooks Road, Rome, NY 13441-4114. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
<small>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.</small>				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE APRIL 2001	3. REPORT TYPE AND DATES COVERED Final Apr 97 - Nov 98		
4. TITLE AND SUBTITLE UPDATE ANALYSIS IMPLEMENTATION (UAI) ADVANCED TECHNICAL PROTOTYPE (ATP)		5. FUNDING NUMBERS C - F30602-95-C-0292 PE - 63726F PR - 2810 TA - 01 WU - 60		
6. AUTHOR(S) J. Huff and S. Ochsner				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) PRC, Inc. 1410 Wall Street Bellvue NE 68005		8. PERFORMING ORGANIZATION REPORT NUMBER  N/A		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFED 32 Brooks Road Rome NY 13441-4114		10. SPONSORING/MONITORING AGENCY REPORT NUMBER  AFRL-IF-RS-TR-2001-67		
11. SUPPLEMENTARY NOTES Air Force Research Laboratory Project Engineer: Patrick K. McCabe/IFED/(315) 330-3197				
12a. DISTRIBUTION AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) The goal of the Update Analysis Implementation ATP is the integration of commercial off-the-shelf (COTS) and Government off-the-shelf (GOTS) software that will provide for the simultaneous and transparent update of multiple heterogeneous databases. The point of departure for the UAI ATP was the Query Support Processor (QSP) system developed for Rome Laboratory under a previous effort.				
14. SUBJECT TERMS Heterogeneous Database Access and Update, Schema Management, Virtual Global Schema			15. NUMBER OF PAGES 32	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

## TABLE OF CONTENTS

	<u>Page</u>
SECTION 1. INTRODUCTION	1
1.1 Purpose	1
1.2 Background	1
1.2.1 The Query Support Processor (QSP)	1
1.2.2 Update Analysis Implementation	3
1.2.3 Update Analysis Phase I Prototype	4
1.2.4 Update Analysis Phase II Prototype	6
SECTION 2. UAI ATP ENVIRONMENT	8
2.1 System Overview	8
2.2 JADE User Interface	8
2.3 Request Routing	9
2.4 Request Execution	10
2.5 Schema Repository Manager	11
2.6 UAI Administration Tools	12
2.7 UAI Interprocess Communication	12
SECTION 3. OPERATIONAL ANALYSIS	13
3.1 Addressing System Limitations	13
3.2 Developments that Impact Data Kinetix	13
SECTION 4. DATA KINETIX VERSION 2 TECHNICAL OBJECTIVES	15
4.1 Primary Objectives	15
4.2 Other Objectives	15
SECTION 5. DATA KINETIX VERSION 2 TECHNICAL APPROACH AND ARCHITECTURE	17
5.1 Use of CORBA	17
5.2 Virtual Fusion	17
5.2.1 Virtual Fusion Example	18
5.3 Schema Repository Redesign	21
5.4 Standard Application Program Interfaces	23

## LIST OF FIGURES

<u>Figure</u>		<u>Page</u>
1	UAI System	2
2	Request Routing Process	10
3	UAI ATD MetaSchema Repository Schema Diagram	11
4	Version 2 Architecture	17
5	Virtual Fusion Example	19
6	Row Mapping	20
7	Example Result	21
8	DK MetaSchema Repository Schema Diagram	22

## **SECTION 1. INTRODUCTION**

### **1.1 Purpose**

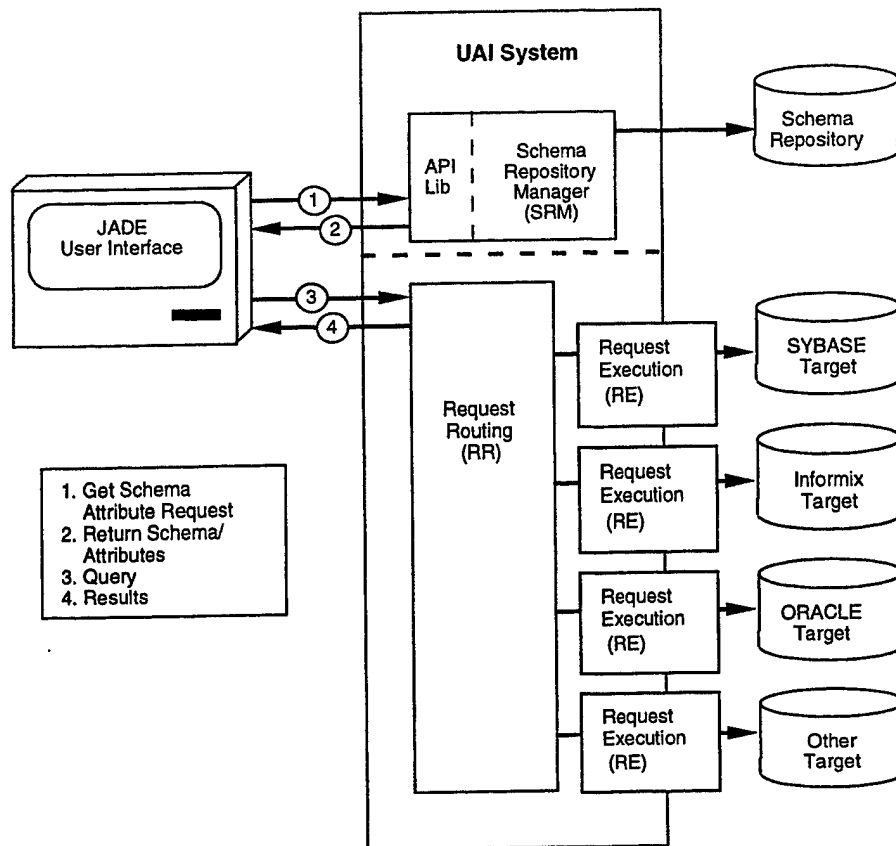
This Update Analysis Implementation (UAI) Advanced Technical Prototype (ATP) Final Technical Report is submitted to Air Force Rome Laboratory (AFRL) in compliance with contract F30602-95-C-0292. This report provides an understanding of the functional capabilities delivered under the UAI contract, lessons learned from the effort, and a description of the next iteration of the capability as Data Kinetix (DK) middleware under the Databases for the 21<sup>st</sup> Century program.

### **1.2 Background**

The goal of the Update Analysis Implementation ATP is the integration of commercial off-the-shelf (COTS) and Government off-the-shelf (GOTS) software that will provide for the simultaneous and transparent update of multiple heterogeneous databases. The point of departure for the UAI ATP was the Query Support Processor (QSP) system developed for Rome Laboratory under a previous effort.

#### **1.2.1 The Query Support Processor (QSP)**

Update Analysis is based on the earlier prototype for the Query Support Processor. QSP provided a framework for addressing many of the problems of heterogeneous database access through the use of an entity-relationship model schema repository. The schema repository contained a description of network schemas at both the physical and virtual level. In the QSP prototype, each data source available to the prototype's query mechanism was described in the schema repository in its true physical organization. This meta data was overlaid with a second description of data entities organized into topics of related information without regard for their physical location or data representation. This virtual name space, shown in Figure 1, UAI System, accomplished a very important first step towards helping the user to deal with the vast amount of data available on a network by providing a logical organization of related data. Additionally, the use of a virtual name space solved the immediate problems of combining data from two sources, resolving synonym and homonym confliction.



**Figure 1, UAI System**

Homonym conflict is the result of two entities which have the same physical name but represent incompatible data, such as ID which refers to system identification in one table and ID which refers to user identification in another table. These two fields obviously refer to different information and within the QSP schema repository are correctly represented as two distinct logical entities, preferably with descriptive names such as SYSTEM\_ID and USER\_ID. This allows the user to select the appropriate field for a query. The definition of a homonym conflict is two entities that are incompatible in data type and context but have the same physical names.

Synonym conflict is most common where databases have expanded their area of interest and overlapped, or when the same data is stored in two different databases to support different applications, sets of users, or performance requirements. The naming conventions may be different based upon the administrative policy or the nature of the database management system. A common example is case sensitivity. Over time, many of these data entity names will be standardized by national agencies, but in many legacy systems, and in differing versions of legacy systems that are concurrently active, synonym conflict is common.

Following the development of the original QSP, the system limitations and lessons learned were noted and many were resolved during the Update Analysis effort. One limitation of QSP was the lack of true virtual data views. In the QSP approach, elements from two sources could be combined in a logical data view, but the user was required to supply join information as part of a query against the view. A second limitation was the inability to deal with differing data representations (such as unit of measure) of the same data in different sources. It was also noted during the original QSP effort that a mechanism for automatically loading and keeping the information in the schema repository synchronized with the physical database schemas would be needed in future versions. And finally, the original focus of QSP was heterogeneous queries. Since it was designed to be a query only system, another effort would be required to enhance the system to support updates.

### **1.2.2 Update Analysis Implementation**

The Update Analysis Implementation prototype continued in the direction established by QSP, resolving some of the issues that remained open at the conclusion of the QSP effort to provide a more stable environment for multiple data source access and providing a multiple target update capability. Our approach to implementing the Update Analysis was based on a two-phase life cycle.

The Phase I Prototype for Update Analysis provided a stabilized baseline of data access functions including the communications process, the DBMS isolation layer, the repository access mechanism, and the request mechanism. Some of the capabilities demonstrated for the Phase I Prototype included true logical data views, homonym and synonym resolution, unit conversion with precision, datatype conversion, automatic generation of transaction strategies, storage and reporting of historical transaction performance data, an ad hoc query interface, and a peer-to-peer communication substrate.

The Phase II Prototype demonstrated multiple database update capabilities. Three types of multiple database updates are included: replicated update, global single entity update, and update entity by view. The Update Release Queue application demonstrates how multi-target updates can be displayed to a release authority for approval before execution. The Automated Schema Synchronization feature demonstrates how the Update Analysis Implementation Prototype can keep its schema repository current in a low-maintenance environment by automated query of target schemas and reporting discrepancies to the administrator. This allows the prototype to remain viable in a volatile data environment without forcing local administrators to post their schema changes to the repository.

Enhancements to the schema repository table structure took the next step towards removing the physical boundaries that make it difficult to compare data from one source to another by creating virtual data views based on implied joins. Data views could be created by a knowledgeable administrator using SQL "where" clauses to support the



view. These "where" clauses were then stored in the repository as part of the view definition. The SQL "where" clauses were then appended to a user's query of the view transparently and incorporated into the subquery build process. However, simple SQL statements could not solve all of the incompatibility problems between two sources that could arise based on differences in unit of measure and data type. As a result, the SQL statements had to be augmented with additional functions to obtain the necessary conversion of data to a common representation that could be compared during the join process.

### 1.2.3 Update Analysis Phase I Prototype

Many of the preliminary requirements of the UAI prototype were satisfied to some degree by the enhancement of extensions to QSP that were developed under a PRC IRAD effort. Some of these requirements reflect inputs from users and the Government after the QSP final demonstration. Other requirements were suggested by PRC based on our own growing understanding of the problems of presenting a unified view of data to the user. An examples are the requirements for virtual views and synonym and homonym deconfliction. The thrust of these requirements is aimed at resolving difficulties in combining federated databases into one logical database discovered during the QSP effort.

The original QSP prototype provided the preliminary framework for addressing many of these problems through the use of an entity-relationship model for the schema repository. The user organized data entities in whatever fashion made the most sense, regardless of their true location and organization following a concept of *virtual data views*. The repository resolved the location and true name of the elements of a user query and generated subqueries against source databases accordingly. QSP automatically recognized user queries that contained data from separate tables and performed user-specified joins to effect the creation of a logical table. The intermediate solution was to present the data entities to the user according to their physical organization but with logical names. The end user had to create the logical view manually by creating cross database joins with key elements for comparison.

The Update Analysis Phase I prototype provided an improved solution which was to allow the administrator to build a virtual view based on implied joins which were stored in the repository and added to an user query against the view. This way, a user sees a collection of related data elements and creates a standard query. The query is submitted to the parser, which recognizes that the elements of this view represent one or more tables and appends the implied join sql to the user query. This approach shielded the user from the very complex process of mapping data entities from one database to another, and provided a truly logical organization of the network data so that it made more sense and was easier to navigate.

Synonym conflict is most common where databases have expanded their area of interest and overlapped, or when data is replicated in two different databases. The naming conventions may be different based upon the administrative policy or the nature of the database management system. A common example is case sensitivity.

Another example of conflicting entity names leads to the requirement for a data conversion capability. Homonym conflict is the result of two entities which have the same physical name but represent incompatible data, such as ID which refers to system identification in one table and ID which refers to user identification in another table. These two fields obviously refer to different information and within the QSP schema repository are correctly represented as two distinct logical entities, preferably with descriptive names such as `SYSTEM_ID` and `USER_ID`. This allows the user to select the appropriate field for a query. The definition of a homonym conflict is two entities that are incompatible in data type and context but have the same physical names.

The homonym deconfliction solution has side effects however, which were discovered as the prototype was presented to the users. Sometimes, homonyms are really synonyms. For instance, two databases contain the field "latitude", however in one database, latitude is a six-character field and in another it is a seven-character field. The reason for the longer field is that one database has concatenated a hemisphere designation onto the latitude field. Obviously direct comparisons of the two fields will fail because they have incompatible formats. The original QSP prototype was capable of recognizing this situation, and since it could not join a seven-character field to a six-character field, the two physical fields were represented by two logical fields, `LAT_HEMIS`, and `LATITUDE`. However, there is a case for a query that joins the fields. The requirement is for a conversion function to translate the format. A simpler example of homonyms which are synonyms in disguise is ship length which is recorded as feet in one databases and meters in another. Obviously, joins between feet and meters are nonsensical - the data must first be converted to a common unit to be joined.

Under Independent Research and Development, extensions were developed to solve this problem for the unit conversion issue. To do this, developers took advantage of the entity-relationship organization of the schema and added unit type as an attribute on the relationship between data elements and databases. The target fields can still be represented as one virtual entity to the user, but the IRAD QSP extension recognizes that the data is stored differently in the results of individual subqueries. When a user generates a query that requires entities with unit type attributes to be combined, the data is converted from one unit type to another as the final result set is compiled. The conversion algorithm is stored in a file that is named according to the "from" and "to" specifications of the unit designations, e.g., "feet\_to\_meters". This handles simple arithmetic or datatype conversions. It cannot, however, handle field concatenation or complex data mapping problems. (This level of complexity is more applicable to a data exchange capability than a simple, multi-database retrieval function, and it will be

accomplished in the combination of data exchange and data retrieval capabilities for the next version of the capability (see section 3)).

The UAI Phase I effort focused on integrating the capabilities developed under QSP with the IRAD work and stabilizing the performance and error checking as a baseline to begin the actual update implementation for Phase II. The enhancement work in the area of the repository architecture during this first portion of the effort was accomplished with a view towards later incorporation of the update parsing information and data, and processing modeling information.

#### **1.2.4 Update Analysis Phase II Prototype**

The Phase II Update Analysis Prototype focused on three types of updates, the replicated update, the global single entity update, and the update entity by view.

- The *replicated update*, updates multiple replicated tables which occur in separate databases on the network. This is a straightforward implementation that mimics a single database update with the added safeguards of multi-phase commit and rollback to insure the synchronism of the targets is maintained. This type of update is only useful to environments that have a great deal of schema duplication and data replication since the field to be updated and the constraints placed upon the update must occur in every target database.
- The *global single entity update* allows the user to update the value of a single field wherever it exists on the network. The Update Analysis ATP software determines which tables contain the entity and performs all updates accordingly. This type of update does not have a constraining clause. The multi-phase commit and rollback processes are especially critical for this type of update since it is likely to target several tables.
- The *update entity by view capability* is very complex and based upon the concept of data views. Data views represent an assortment of data entities from several different sources with a common focus. An update constrained by the value of an entity, which is in the same logical view but a different physical table, requires an implied subquery. The implied subquery extracts the rows of the first entity that match the constraints of the second *and* matches join keys. A second transaction performs the update against the results of the first.

The ability to automatically query for and receive schema information from targets is critical to a successful operational installation. The significance of increasing the autonomy of the network data access and update server has become apparent by our discussions with users and other contractors during the development of the QSP

prototype. All vendors who are currently pursuing this type of research are concerned about the ability of a centralized schema repository to maintain concurrency with target databases without the need for intensive database administration. Some of the solutions proposed simplify the problem technically, but show little understanding of the dynamics of an environment of diverse federated systems with user and administrative groups who do not want to trade network data access for a loss of control over their own source of data. The Sybase solution, for example, requires all network database administration to be accomplished through the network data access server. While this method would work in theory, it is impractical from an operational standpoint where database administrators want to continue to manage their own systems through the tools that they have been trained to use. The Update Analysis ATP solution provides a process that periodically queries each defined data source for its meta data. This information is compared to the meta data stored in the schema repository, and any discrepancies are flagged and presented to an administrator for resolution. The Metaschema Autosynchronization Process can be used to initially load the physical schema as well as looking for changes and recommending updates.

## **SECTION 2. UAI ATP ENVIRONMENT**

### **2.1 System Overview**

The Update Analysis Implementation Advanced Technical Prototype provides an Application Programming Interface to support the transparent retrieval and update of data resident on connected data servers. Queries for composite data result sets are built from topical data views that combine relevant data entities independent of the physical organization or location of the data. Updates against these views are translated to updates at each source, constrained by the source's administrative controls. The UAI ATP provides access to data through its ad hoc query interface, or it can be used as database access middleware that supports application or tool access to multiple data sources via its Application Program Interface (API).

The UAI ATP provides a method by which a user or an application may view or update the data stored in multiple heterogeneous databases as though it were integrated into a single logical database. This logical database can be queried or updated by the user via the Java Ad hoc Data Environment (JADE) user interface, or via the DK Application Program Interface (API). This API exists in both C and Java versions. The UAI ATP middleware accepts virtual queries from JADE, or through the API, and processes them into subtasks that when executed, retrieves data from all databases that are required to satisfy the original virtual query. Updates are processed with some constraints. Only privileged users are permitted to request updates through UAI and all update requests are held as Data Change Requests (DCRs) to be reviewed and approved by a responsible producer before they are permitted to execute. The primary components of UAI are the JADE user interface, the Request Routing module, Request Execution module and the Schema Repository Manager. UAI also provides a set of repository administration tools. This architecture is depicted in Figure 1, UAI System, and described in the following paragraphs.

### **2.2 JADE User Interface**

The purpose of the JADE User Interface is to allow a user to generate queries and updates directly against the target databases without an intervening application. It is also a useful tool for browsing the network database environment because it displays the user's view of the data environment as though all the data were contained in a single database. There is a point and click SQL generation window that guides the user in building queries against their view.

The JADE User Interface displays the network data environment to the user in a virtual organization that is independent of the physical scheme found in the target DBMS's. Data elements are grouped into virtual tables or topics based on the classification scheme. A simple interface leads the user through the selection of data elements and associated

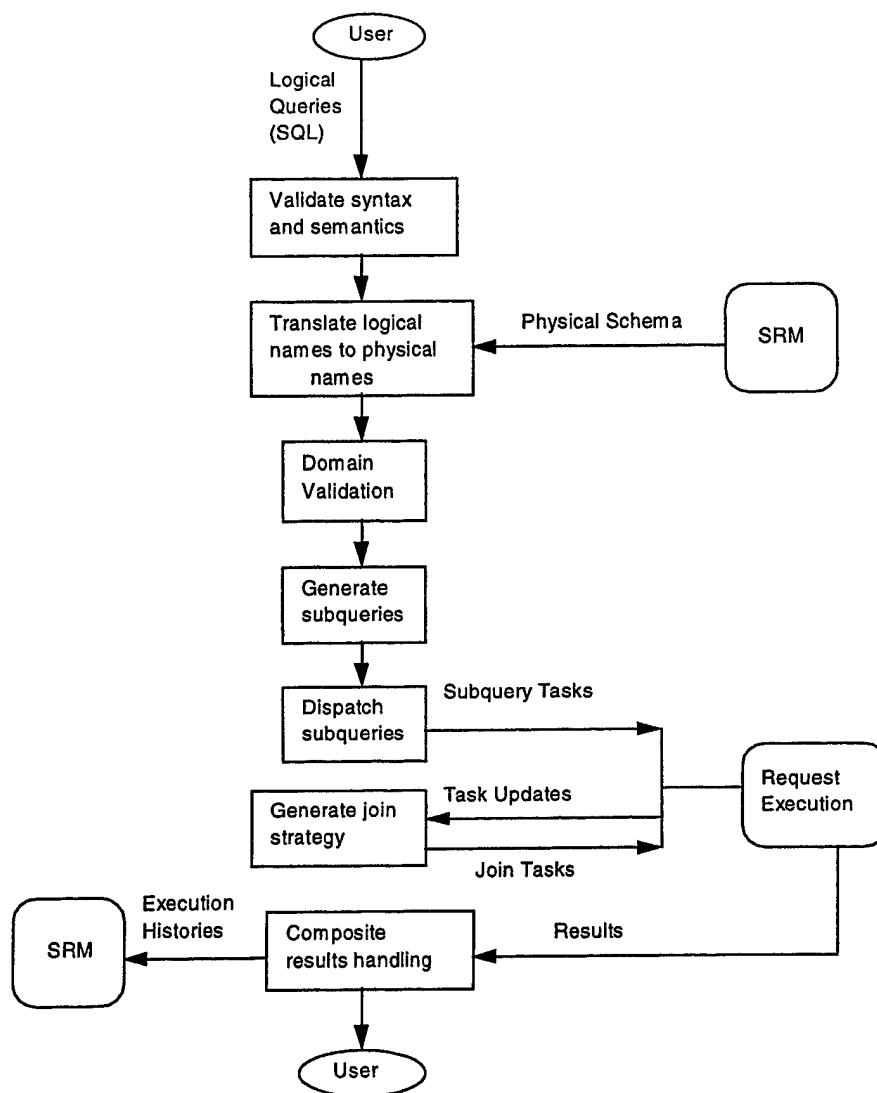
information to query or update elements from a particular topic or group of topics. As the user enters data, an SQL statement is generated. Advanced users may enter SQL queries into a pop-up text window directly. When the query has been composed, the user transmits it, and the query name appears in the status window. When the status window registers the query as complete, the user may view the results.

Input to the JADE User Interface is the data entered by the user, which JADE translates into repository requests or query/update statements. Repository requests, including the GetSchema Request, GetAttributes Request and ValExec Request, are function calls to the DK Interface Library. The library transmits the requests to the Schema Repository Manager Module or to the Request Routing Module.

JADE receives virtual network schema information (Views) and attributes on data elements (Entity Attributes) from the Schema Repository Manager, and receives result data (Response Data) filtered through the Request Routing Module, and outputs this information to the display.

### **2.3 Request Routing**

The Request Routing (ReqRout) Function receives SQL statements and parses them to perform request validation, virtual to physical element name mapping, SQL augmentation to affect transparent cross-database joins, subquery generation, and task routing. The UAI software then develops optimized cross database join strategies, and receives composite result sets from the target nodes which it transmits back to the client. The request routing process is shown below in Figure 2, Request Routing Process.



**Figure 2, Request Routing Process**

## 2.4 Request Execution

The Request Execution and Data Access components handle the execution of queries against each target database. In the optimal system configuration, a Request Execution/Data Access module should exist on each remote DBMS node. This prevents large amounts of data from being transported across the network unnecessarily. If a particular subquery has a large result set, its node becomes the host for the join and gathers smaller results to it to form a reduced composite result.

The queries received by Request Execution/Data Access are in the form of Task commands sent out by Request Routing. Request Execution/Data Access has five service

## 2.5 Schema Repository Manager

[illegible]

**Figure 3, UAI ATD MetaSchema Repository Schema Diagram**



## **2.6 UAI Administration Tools**

A set of administration tools is provided with the UAI middleware system to help manage the schema repository. These tools are used to define and administer the virtual database, load new schema, provide enhanced dictionary information and to keep the repository synchronized to the network's database environment. An automated schema synchronization process is included that detects and helps resolve changes at the physical database level.

## **2.7 UAI Interprocess Communication**

The UAI ATP API provides a non-blocking, multi-threaded mechanism for clients to submit queries against topical views using the windows-based event-driven model. The client application submits a query, the address of a function to be called when results are available, and the address of a function to be called if an error occurs. This interface can also be used to retrieve information from the MetaSchema Repository (MSR).

## **SECTION 3. OPERATIONAL ANALYSIS**

### **3.1 Addressing System Limitations**

The Update Analysis Advanced Technical Prototype (UA/ATP), or Data Kinetix (DK) as it has since been renamed, showed that it is possible to develop a database access middleware that can successfully support applications in a multi-database environment. However, as part of the development and implementation of Data Kinetix Version 1, a number of physical system limitations have been uncovered that will need to be addressed in Version 2. To address these limitations, Data Kinetix Version 2 will need:

- The capability to efficiently handle large results when processing subqueries and generating composite results.
- The ability to assemble query results that contain both relational and non-relational data to include mechanisms for defining more complex relationships.
- An architecture that can be easily integrated with other heterogeneous access products, provides standard application interfaces, and is DII/COE compliant.

Data Kinetix Version 1 has experienced some problems with handling very large result sets that can be generated during the execution of subqueries and when compiling composite results from multiple databases. The primary issue is the time it can take to complete cross-database joins by sending subquery results from one database to another, load the subquery results and re-issue a query to complete the join. The design for Data Kinetix Version 2 will solve this problem using a different and much more efficient approach for joining data from multiple databases and paging results on a demand basis. This method, referred to as Virtual Joins, is described in section 4.

### **3.2 Developments that Impact Data Kinetix**

The data that intelligence analysts require access to is changing to encompass a more global view of information. This information is not usually available from a single source or database and often includes relational data, images, graphics, sound, and video. To accommodate this change, Data Kinetix will need to be enhanced to become more object oriented so that data of different types can be more easily related and accessed. In addition, more sophisticated approaches to data fusion will need to be developed in order to present meaningful views of information from diverse sources. For these reasons, Data Kinetix Version 2 will include a redesigned schema repository structure that will better support the definition of relationships between diverse data. This new structure will also improve performance when retrieving meta data from the schema repository.

Over the course of the last few years, more DBMS vendors have developed products that provide heterogeneous database access. There are also several Government-developed capabilities available. In addition, the call level interfaces for SQL based systems have seen some standardization through interfaces like ODBC and JDBC. It has therefore become important for Data Kinetix to become more specialized by focusing on the data fusion problem. Data Kinetix Version 2 will become more flexible in its architecture so that it can integrate well with other heterogeneous access products. Since fusion of data is a separate and unique problem, DK will be able to use any heterogeneous database access capability to provide data to its fusion module.

## **SECTION 4. DATA KINETIX VERSION 2 TECHNICAL OBJECTIVES**

### **4.1 Primary Objectives**

The technical objectives for Data Kinetix Version 2 are focused on the three areas that were identified as being problematic in Version 1 (see section 3). These objectives are to provide a more efficient data fusion capability, support data objects, and provide standard interfaces. In order to meet these objectives, we have decided on an architecture that uses CORBA as the framework for communication with client applications and a given heterogeneous database access product.

### **4.2 Other Objectives**

Other specific objectives are:

- The Virtual Data Fusion capability of Data Kinetix middleware will perform cross-database joins in memory prior to extracting the required rows specified by each subquery.
- The Virtual Data Fusion capability will extract data from each database a page at a time, and return data to the client a page at a time. Page size will be configurable. Clients can request all pages be sent one after the other or on a per request basis.
- The schema repository structure will be redesigned to eliminate structures that support extra levels of metadata that are no longer required. New structures will be added to support the definition of relationships to non-relational and non-database information. The entire repository structure will be further enhanced to improve performance.
- Provide client applications with a standard Application Programming Interface (API) for communication with Data Kinetix to include:
  - JDBC interface for Java applications.
  - An ODBC interface for non-java applications.
- The JDBC and ODBC interfaces will be interchangeable with standard JDBC and ODBC for direct database connections. An application connected to this system can use a standard JDBC or ODBC connection with no loss of functionality.
- There will be a CORBA API available for applications to use directly.
- CORBA will be used for inter-process communications between Data Kinetix services.

- The JDBC interface will be packaged so that java applets can be connected with no special configuration using the ORB included with Netscape Navigator 4.X.
- The access mechanisms will not be DBMS specific.
- The access mechanism will provide access to multiple databases using commercial or Government products for heterogeneous database access.

## SECTION 5. DATA KINETIX VERSION 2 TECHNICAL APPROACH AND ARCHITECTURE

### 5.1 Use of CORBA

The objectives described in section 4 will be achieved in this effort by building CORBA communications into the Data Kinetix middleware product. Applications will effectively communicate with the database using a set of CORBA services in DK to parse virtual queries, issue subqueries and fuse subquery results. Figure 4 shows the architecture of the Data Kinetix Version 2 system.

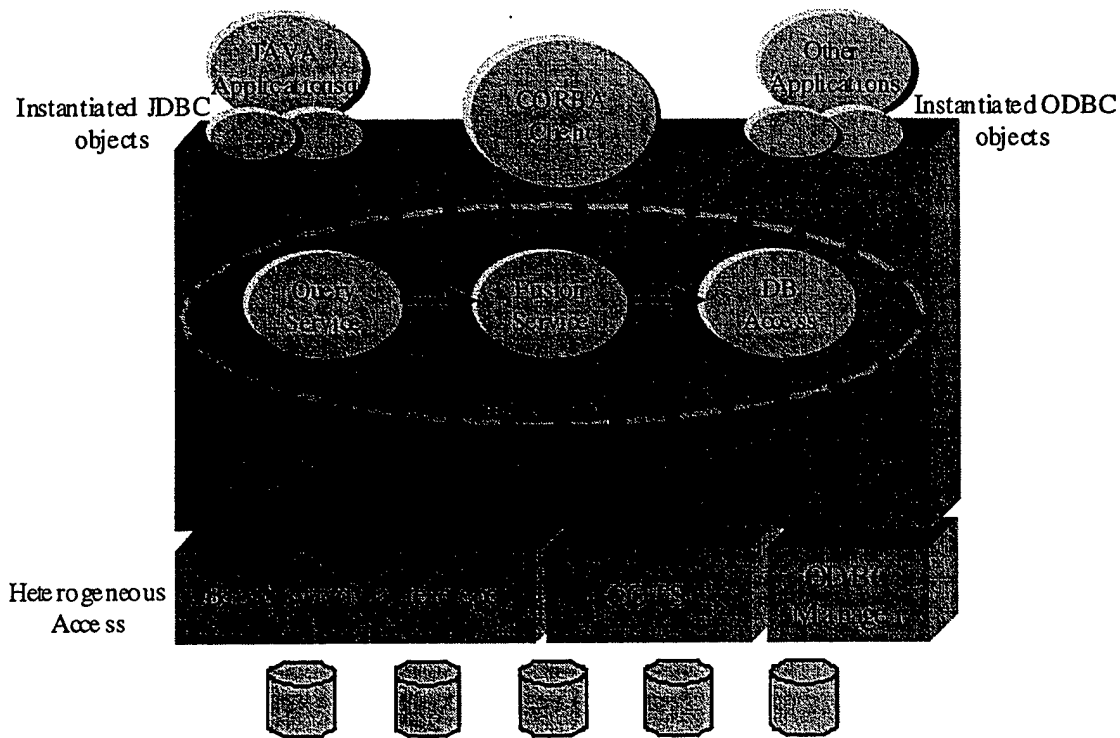


Figure 4, Version 2 Architecture

### 5.2 Virtual Fusion

The Virtual Fusion capability in Data Kinetix Version 2 will replace the current method employed by Version 1 and will greatly reduce the amount of data that needs to be moved in order to create a composite result. In DK Version 1, a separate Query Execution Module (QEX) was installed on each database node that was part of the DK

configuration. Each QEX was responsible for executing its tasked subquery, reporting its result size back to Query Status, and then either sending its results to another QEX or receiving results from other QEX processes. The receiving QEX would then create a temporary table for each result, load the results, and then re-query the database to join all results together. This method, although reliable, could take a significant amount of time when result sizes were large, and the entire process would have to complete before the user would see any results returned. This approach also required that DK be granted special privileges in each DBMS so that it could create temp tables and load data. The overall architecture of DK also required that a module be installed on each database node.

With Virtual Fusion, all result processing will be done from a single module. Instead of sending entire subqueries to each node, the VF module will query for only the required keys. The key information will be compared in memory outside any DBMS environment and a set of Link Vectors will be produced. These link vectors will then be used in the creation of a virtual image that describes the entire composite result. Then, the VF module will use the vectors to retrieve the required rows from each database a page at a time. The advantages of this approach are that as soon as the first virtual page is generated in memory, it can be retrieved and returned to the user allowing the user to see results almost immediately regardless of the actual size of the results. Since results are handled and processed a page at a time, system and network resources won't be over taxed. This approach also simplifies the DK architecture so that in Version 2, a separate module will not have to be installed on each database node and the DK system will not need to be granted any special privileges for creating temp tables and loading them. DK will also be able to query any database it can "see" without the need to install new modules or do any extensive reconfiguration.

### **5.2.1 Virtual Fusion Example**

The following example shows, in more detail, how the Virtual Fusion capability will work.

- The virtual query example in Figure 5 requests data from two databases (MIDB and EWIR).
- The Join in the query specifies key or keys from both databases (ELNOT and ELINT\_NOTE).

```

select  FQL.BE_NUMBER, FQL.EQUIP_CODE,
        S03.ELNOT, S03.TREE_NUMBER
from    FQL, S03
where   FQL.ELINT_NOTE = S03.ELNOT
order by FQL.EQUIP_CODE

```

### **Schema: 1.0**

#### **FQL**

BE_NUMBER
EQUIP_CODE
EQUIP_ID_NUMBER
LOCATION_TYPE
LAT_LONG
ELINT_NOTE

### **Schema: EWIR**

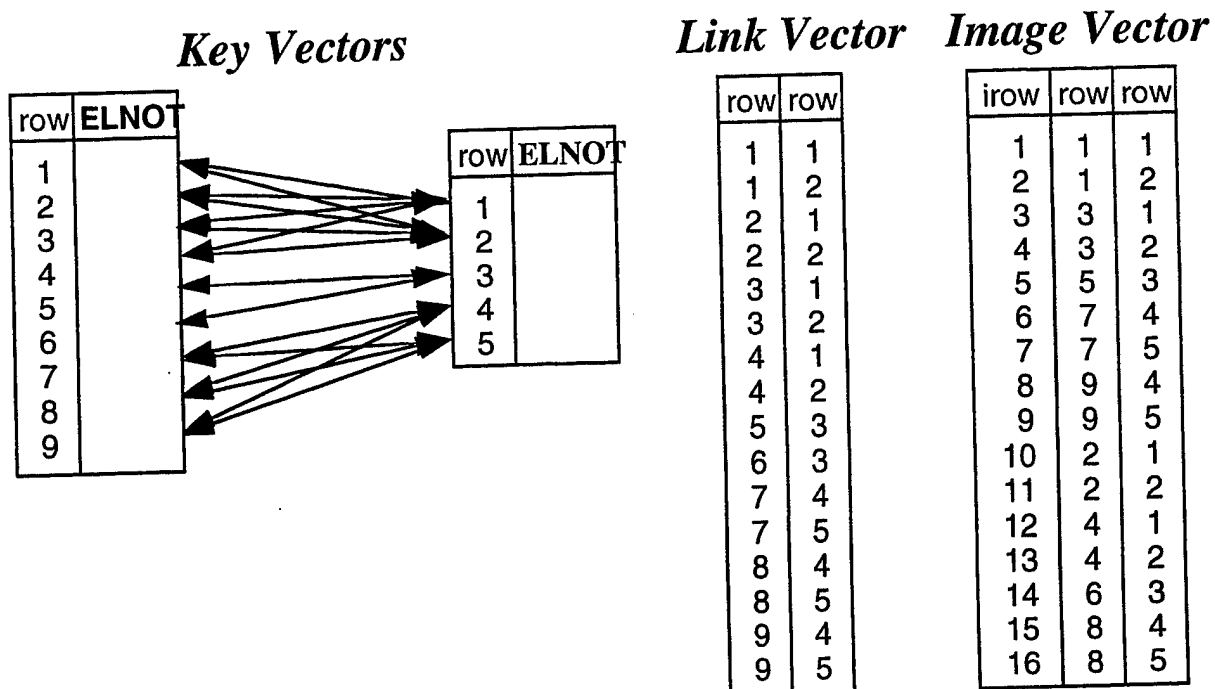
#### **S03**

ELNOT
TREE_NUMBER
FORMAT_
SUFFIX_CODE
TEXT

**Figure 5, Virtual Fusion Example**

- The subqueries generated for each database retrieve only the required keys.
- Key comparison is done in module memory to produce a set of link vectors.
- Link vectors are used to create a set of image vectors that map the virtual image row to individual rows from each database as in the following figure.





**Figure 6, Row Mapping**

- Virtual image rows are generated on demand; physical rows are retrieved only as needed.
- Virtual images can be paged for better management of huge composite results:
  - Eliminates the need to retrieve all data at once
  - Only the rows needed to produce a virtual page are ever retrieved

## Virtual Image

irow	row	row
6	7	4
7	7	5
8	9	4
9	9	5
10	2	1
11	2	2

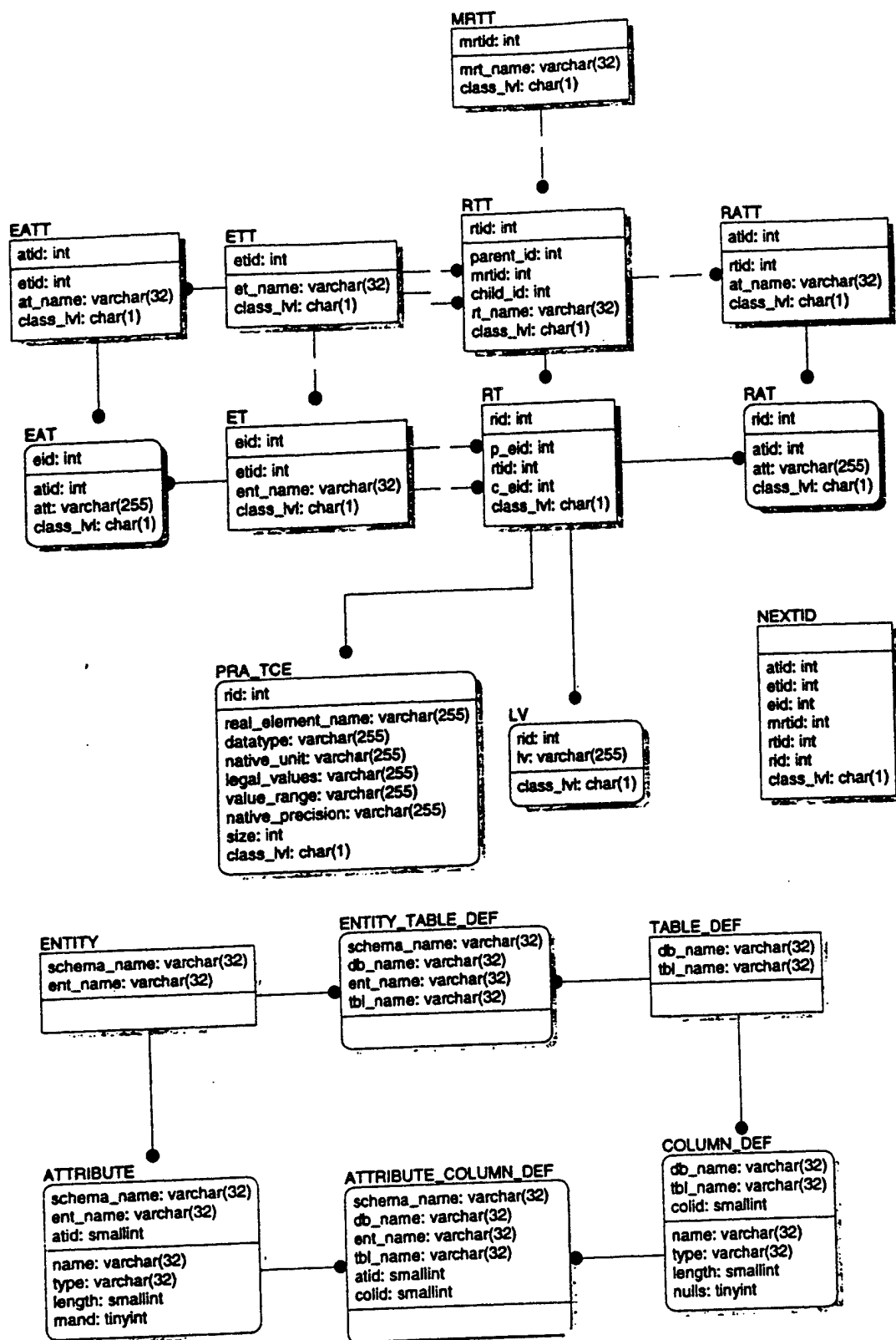
BE_NBR	EQPCD	ELNOT	TREE_NBR

Figure 7, Example Result

### 5.3 Schema Repository Redesign

In Data Kinetix Version 2, the schema repository will undergo a redesign to improve the efficiency of queries for metaschema information. The redesign will reduce some of the complexity that was in the Version 1 structure, but in so doing will also eliminate some of the extensibility. The loss of extensibility is primarily in the capability to add new entity types and relationship types dynamically. In the Version 2 schema, to add new types, the table structure of the schema repository will have to be updated. This is not a critical loss in capability since once defined, it does not need to change anyway. The trade-off of improved performance over loss of extensibility will be worthwhile.

In the Version 2 schema we have created separate tables to manage information about physical schema directly. For example there is now a table that describes databases, a table for information about tables, and a table for element descriptions. Key set information is now stored separately as well. New tables have been added to manage views and queries against views. There is also a set of tables that describe schema mapping that will be used for product generation. These tables are used to describe how elements from one database map to elements of another database.



1, 1 / 1, 1 - 12:58:18 PM, 4/6/2001

Figure 8, DK Metaschema Repository Schema Diagram shows the layout of the redesigned repository.

## **5.4 Standard Application Program Interfaces**

The Data Kinetix Version 2 API will follow the JDBC and ODBC standards so that any application that adheres to these standards can easily communicate with Data Kinetix. This also provides applications the advantage of being completely independent of any specific data access mechanism. Applications can also connect to a given database directly using standard JDBC or ODBC drivers, although the application would be giving up the option to obtain fused data from DK by doing so. However, standard JDBC/ODBC drivers and DK JDBC/ODBC drivers can coexist within the same application. Raw CORBA methods will also be available to applications. The full functionality of the system will be accessible using any of these interfaces.

***MISSION  
OF  
AFRL/INFORMATION DIRECTORATE (IF)***

*The advancement and application of Information Systems Science  
and Technology to meet Air Force unique requirements for  
Information Dominance and its transition to aerospace systems to  
meet Air Force needs.*